Verschiedenes

Jörg Faschingbauer

November 16, 2020

Table of Contents

FASCHINGBAUER

Overview

FASCHINGBAUER

Overview

FASCHINGBAUER

GDB — Overview

FASCHINGBAUER

- Source level debugger
- Commandline *only*
- GUIs are frontends (`stdin`, `stdout`, `stderr`)
- All features a debugger can have (breakpoints, watchpoints, threads, ...)
- Can output arbitrary expressions
- Toolchain: use the options `-g` and `-O0`

Starting the Debugger

FASCHINGBAUER

- Start a program under debugging (main use)
- Attach to running process
- “Post mortem” debugging: core files

Overview

FASCHINGBAUER

Simple Example

FASCHINGBAUER

`ls -dl .` in the debugger ...

GDB with `ls -dl .`

```
$ gdb ls
...
(gdb) run -dl .
Starting program: /bin/ls -dl .
[Thread debugging using libthread_db enabled]
drwxr-xr-x 8 jfasch jfasch 16384 Feb  8 18:32 .
[Inferior 1 (process 30876) exited normally]
(gdb)
```

What Else?

FASCHINGBAUER

Basic commands for newbies:

- Quit: `quit` or Strg + d
- Global help: `help`
- Help for specific command: e.g. `help run`
- Topic-help: e.g. `help running`, `help breakpoints`

→ Nobdy has to write out entire command, unique command prefix is sufficient

→ History like Bash (through `libreadline`)

Movement Inside the Programm

FASCHINGBAUER

- Strg + c : interrupt
- `continue`: ...
- `backtrace`, `bt`: view call stack
- `step`: next line; step into when function
- `next`: next line; don't step into function
- `finish`: continue until end of current function, halt afterwards
- `print`: show value of expression

Attaching to a Running Process

FASCHINGBAUER

Running process → Process ID (PID)

GDB mit `evince`

```
$ ps -u jfasch|grep -w evince
  753 pts/1    00:00:00 evince
$ gdb -p 753
... Symbole lesen ...
(gdb)
```

- Process is suspended → `continue`
- Quit debugger `detach` → Debugger **de**tached

Overview

FASCHINGBAUER

Core Dumps

FASCHINGBAUER

"Core Dump"

- Process status written in a file, debugger can reproduce status
- Written by kernel when letal signals are received (`SIGSEGV`, `SIGBUS`, `SIGFPE`, ...). Kurz: **Bugs**!
- File `core` in process's Current Working Directory (configurable through `/proc/sys/kernel/core_pattern`)

Core Dumps - Limits (1)

FASCHINGBAUER

- Kernel enforces per-process linits: memory usage, core size, ...
- Core size mostly limited to 0 bytes → no core dump
- Limits are inherited
- → Configuration per user or global
  - `~/.bashrc`: per user
  - `/etc/profile`: global

Core Dumps - Limits (2)

FASCHINGBAUER

Increasing core limit

```
$ ulimit -a
...
core file size          (blocks, -c) 0
...
$ ulimit -c
0
$ ulimit -c unlimited
$ ulimit -c
unlimited
```

Core Dumps in Action

FASCHINGBAUER

Buggy Program

```
$ ./buggy
Segmentation fault (core dumped)
$ ls -l core
-rw------- 1 jfasch jfasch 241664 Feb  9 15:20 core
$ file core
core: ELF 64-bit LSB core file x86-64, version 1 (SYSV
$ gdb ./buggy core
```

- From now on almost all debugger command are usable
- Except for `next`, `continue`, ...

Core Dumps: Miscellaneous

FASCHINGBAUER

- Making your colleagues upset: `kill -SEGV 5432`
- Generating core file from inside the debugger: `generate-core-file` (kurz `gcore`)

Overview

FASCHINGBAUER

Showing the Stack

FASCHINGBAUER

Program interrupted → ready for inspection

- What is our current position?
- What was the call chain to current position?
- → "Stack frames"

Showing the stack

```
(gdb) backtrace
#0  0x00007fa8cc9096f3 in poll () from /lib64/libc.so.
#1  0x00007fa8cce8c6e4 in g_main_context_iterate.clone
#2  0x00007fa8cce8cff2 in g_main_loop_run () from /usr
#3  0x00007fa8cedfadb7 in gtk_main () from /usr/lib64/
#4  0x000000000043e3a4 in main ()
```

Navigating the Stack

FASCHINGBAUER

- Backtrace inside-out → frame #0 is current.

Navigation commands:

- `frame 4`: positioning to frame #4
- `info frame`: more information about current frame
- `up`: one frame "up → backward in history
- `down`: one frame "down → forward in history
- `return`: terminate frame → Caution!
- `return 42`: ... with return value
- `finish`: continue until end of current function, halt afterwards

Overview

FASCHINGBAUER

GDB and the Toolchain

FASCHINGBAUER

Debugging in the darkness is no fun → switch on debugging information, `gcc -g`

- Assigns binary code to source code (file, line, ...)
- Variable names, function names, ...

Switch off optimization using `-O0` → no annoying jumps in the code ...

- Loop-Unrolling
- Block-Reordering
- Variable elimination (register)
- ... whatever the compiler does

GDB and the Toolchain (2)

FASCHINGBAUER

C++: Linker generates code → `-g -O0` during linking as well!

Compiler and linker are often hidden in build systems

- Automake/Autoconf: `configure;make;make install`
  - `configure CFLAGS="-g -O0" CXXFLAGS="-g -O0" LDFLAGS="-g -O0"`
- CMake
  - `cmake -DCMAKE_BUILD_TYPE=Debug ...`

Source Files (1)

FASCHINGBAUER

Problem:

1. Development is done in `/home/jfasch/project/` on my laptop
2. Source and binaries are distributed, e.g. as a `tar`
3. Unpacked in `/tmp/project/`
4. $\rightarrow$ GDB cannot find source file

Solution: mostly awkward ...

Source Files: Searching Along "Source Path" (1)

FASCHINGBAUER

One possible solution: source path. List of directories which could contain source files.

Source Path

```
(gdb) show directories
Source directories searched: $cdir:$cwd
(gdb) directory /tmp/project
(gdb) show directories
Source directories searched: /tmp/project:$cdir:$cwd
(gdb) set directories /tmp/project:/home/ich/project
```

Source Files: Searching Along "Source Path" (2)

FASCHINGBAUER

Problems with "Source Path":

- Source files distributed across subdirectories → each directoy has to be specified explicitly
- Code build from multiple projects almost impossible to handle
- Conflicts with smae filenames in different directories

Source Files: Prefix-Substitution

FASCHINGBAUER

Better solution: Substitution of a path prefix → switching project root

Substitution

```
(gdb) set substitute-path /home/jfasch/project /tmp/pr
(gdb) show substitute-path
List of all source path substitution rules:
  ‘/home/jfasch/project -> ‘/tmp/project’.
```

→ the perfect solution (why so complicated and error prone at first try?)

Overview

FASCHINGBAUER

Viewing Data Using print

FASCHINGBAUER

print Examples

```
(gdb) print i
$2 = 42
(gdb) print 1+1
$3 = 2
(gdb) print ptr->member
(gdb) p strlen("Hey Joe")
```

More information → `info gdb`

Overview

FASCHINGBAUER

Breakpoint Basics

FASCHINGBAUER

Why Breakpoints?

- Code crashes reproducibly, and one wants to inspect code around it
- One doesn't want to single-step, but rather halt at a certain point and single-step from there
- One wants to if code is used altogether
- ... substitute many more reasons here ...

Kinds of Breakpoints

FASCHINGBAUER

Three different kinds of breakpoints:

- *Breakpoints*. Observe code.
- *Watchpoints*. Observe memory and expressions thereof.
- *Catchpoints*. Observe certain kinds of events.

More info → `help breakpoints`

Using Breakpoints (1)

FASCHINGBAUER

Help for the `break` command

```
(gdb) help break
Set breakpoint at specified line or function.
break [LOCATION] [thread THREADNUM] [if CONDITION]
```

- `LOCATION`: position in file, function, address
- `CONDITION`: ...

Setting Breakpoints

FASCHINGBAUER

Breakpoint by position

```
(gdb) break endless.c:7
Breakpoint 1 at 0x4004f5: file endless.c, line 7.
(gdb) break main
Note: breakpoint 1 also set at pc 0x4004f5.
Breakpoint 2 at 0x4004f5: file endless.c, line 9.
(gdb) break endless.c:main
Note: breakpoints 1 and 2 also set at pc 0x4004f5.
Breakpoint 3 at 0x4004f5: file endless.c, line 9.
```

Deleting Breakpoints

FASCHINGBAUER

- `delete`: by breakpoint number
- `clear`: by position

Deleting Breakpoints

```
(gdb) info breakpoints
Num     Type           Disp Enb           What
1       breakpoint     keep y   in main at endless.c:7
2       breakpoint     keep y   in main at endless.c:9
3       breakpoint     keep y   in main at endless.c:9
(gdb) clear endless.c:9
Deleted breakpoints 2 3
(gdb) delete 1
```

Conditional Breakpoints

FASCHINGBAUER

Conditional breakpoint: only hit when a condition is met

Setting a conditional breakpoint

```
(gdb) break 4 if i==0
```

"Break at line 4 of the current file, but only if i == 4"

- Arbitrary expressions
- Condition can be bound to an existing breakpoint, using the `condition` command

Watchpoints

FASCHINGBAUER

Watchpoint: watches variables (in fact entire expression) for change of value

- Extremely helpful when you are searching subtle bugs
- Only applicable to global variables (naturally)
- "conditional" as well
- Is-a breakpoint to gcc $\rightarrow$ `info breakpoints`

Setting a Watchpoint

```
(gdb) watch some_variable
(gdb) watch some_variable > 42
```

Catchpoints

FASCHINGBAUER

Catchpoint: watches certain "events"

- `catch throw`: *throw*ing exceptions
- `catch catch`: *catching* exceptions. Important: *throw*er is visible in the backtrace.

Overview

FASCHINGBAUER

Remote/Cross Debugging

FASCHINGBAUER

Why would somebody want it? → Nobody does!

- Target is not capable enough to run a debugger, and/or hardware is available only there
- Cross debugger on host (PC)
- GDB server on target → minimal setup: 100K
- Debug information can remain on the host
- Communication: serial or TCP

Alternatives:

- Emulation with QEMU/ARM
- Root filesystem via NFS → more (disk) space for development

Remote/Cross Debugging: Target

FASCHINGBAUER

On the target machine: `gdbserver`

- Programm "deployed" as `/bin/bug`
- ... with or without debug information (debug information is only for use with the debugger)

Starting `gdbserver`

```
$ gdbserver --multi :1234
Listening on port 1234
```

- `--multi`: debugger does not stop after program termination

Remote/Cross Debugging: Host

FASCHINGBAUER

Am Host: Cross GDB, E.g. `armv7a-unknown-linux-gnueabi-gdb`

- Ideally sees a binary with debug information
- Started as a *client*

Starting cross gdb client

```
$ armv7a-unknown-linux-gnueabi-gdb ./bug
(gdb) target extended-remote 192.168.1.99:1234
Remote debugging using 192.168.1.99:1234
(gdb) set remote exec-file /bin/bug
(gdb) continue
```

External Debug Information (1)

FASCHINGBAUER

Severe storage restrictions:

- Target has limited flash memory → no space available for debug information
- ... and one cannot boot over network/NFS

Well ...

- Ideally the same binary (bit-wise) should be deployed on target and host
- But: debug info remains on host (when debugging from remote the debug information is not needed on the target)
- Stripped binary goes to host and target

External Debug Information (2)

FASCHINGBAUER

Simplest case: single executable

- Compilation/Linking as always, with debug information (`-g -O0`)
- Externalize debug information

Externalizing debug information

```
$ armv7a-unknown-linux-gnueabi-gcc -O0 -g -o ./bug \
                ./bug.c
$ armv7a-unknown-linux-gnueabi-objcopy \
                --only-keep-debug ./bug ./bug.debug
$ armv7a-unknown-linux-gnueabi-strip ./bug
```

External Debug Information (3)

FASCHINGBAUER

Question: how do I tell the debugger (on the host) where he can find the (externalized) debug information?

Debugger and external debug information

```
$ armv7a-unknown-linux-gnueabi-gdb ./bug
Reading symbols from ./bug (no debugging symbols found
(gdb) symbol-file ./bug.debug
Reading symbols from ./bug.debug ... done.
(gdb) break main
Breakpoint 1 at 0x804855b: file bug.c, line 31.
```

External Debug Information (4)

FASCHINGBAUER

Question: many target shared libraries; do I have to do this for every shared library? → "Linking" the debug information into the binary.

"Linken" of Debug Information

```
$ armv7a-unknown-linux-gnueabi-gcc -O0 -g -o ./bug \
                ./bug.c
$ armv7a-unknown-linux-gnueabi-objcopy \
                --only-keep-debug ./bug ./bug.debug
$ armv7a-unknown-linux-gnueabi-strip ./bug
$ armv7a-unknown-linux-gnueabi-objcopy \
                --add-gnu-debuglink=./bug.debug ./bug
```

External Debug Information (5)

FASCHINGBAUER

Perfect:

- One doesn't have to instruct a debugger anymore
- Careful when using relative paths

More information:

- "Why cross compiling sucks":
  `http://landley.net/aboriginal/documentation.html#why`

More Information

FASCHINGBAUER

- `info gdb`
- `http://www.gnu.org/software/gdb/`

Overview

FASCHINGBAUER

Overview

FASCHINGBAUER

How it Works

FASCHINGBAUER

Interfaces between userspace and kernel:

- System calls ("down")
- Signals ("up")

`strace`: genius (because: simple) tool to snoop on these interfaces

- Debugging
- Understanding
- Hacking
- Headshaking

Basic Usage

FASCHINGBAUER

```
$ strace ls -l
...
```

- Starts the program *under observation* (and terminates with it)
- Attaches to a running program by its PID (`strace -p 1234`)
- Writes all events to `stderr`, or (`-o`) to a log file

Popular Options

FASCHINGBAUER

Options ...

- Filtering: `strace` output can be quite loud
- Following thread/process creation
- More → `man strace`

`-o file`	Protokoll geht nach `file`
`-e fork,execve`	Nur Syscalls `fork` und `execve`
`-e trace=network`	Nur Netzwerk-Systemcalls
`-p PID`	Andocken an laufenden Prozess
`-f`	"Follow"; Folge Child-Prozessen (und Threads)
`-s stringsize`	Argumente bis max. `stringsize` (Default 32)

Example: ssh

FASCHINGBAUER

Which config file are read by ssh?

ssh

```
$ strace -e trace=file ssh server
open("/home/jfasch/.ssh/config", O_RDONLY) = 3
open("/etc/ssh/ssh_config", O_RDONLY)    = 3
open("/home/jfasch/.ssh/server", O_RDONLY) = 4
```

- For anti-manpage-readers: ssh reads ~/.ssh/config, before reading /etc/ssh/ssh_config
- It also reads the private key ~/.ssh/server, as specified in ~/.ssh/config

Example: Skype (1)

FASCHINGBAUER

From the section "headshaking": why is Skype a continuous leader in `top`?

```
$ ps -L $(pidof skype) | wc -l
16
$ strace  -p 11749
futex(0xa81d084, FUTEX_WAIT_PRIVATE ... {0, 49957574} ...
futex(0xa81d068, FUTEX_WAKE_PRIVATE ...
...
```

- *Damn! Polling for a mutex twice a second?!*
- This can be done better, even portably!

Example: Skype (2)

FASCHINGBAUER

Inconceivable nonsense!!

```
$ strace -p 11744
poll([..., fd=7, events=POLLIN, ...], 4, 99) = 0
read(7, 0xa7f0550, 4096) = -1 EAGAIN
read(7, 0xa7f0550, 4096) = -1 EAGAIN
poll([..., fd=7, events=POLLIN, ...], 4, 0) = 0
read(7, 0xa7f0550, 4096) = -1 EAGAIN
read(7, 0xa7f0550, 4096) = -1 EAGAIN
...
```

(`lsof` reveals: file descriptor 7 is a Unix domain socket)

Further Nonsense

FASCHINGBAUER

Software sucks by default ...

- OpenOffice (top star in `top`)
- IBM ClearCase (extremely broken and expensive software)
- (... insert any commercial or ex-commercial software)

Shame: the Skype pattern (`poll()` with a timeout, but reading twice nonetheless) can also be observed in `emacs` — although not in such a tight loop!

Debugging

FASCHINGBAUER

Very helpful for debugging:

- One can take a quick look what the program does
- Debugger is much too heavy to take a quick look
  - Sometimes you cannot stop that program without crashing it
- $\rightarrow$ valuable conclusions
- $\rightarrow$ most problems can be solved without a debugger

Overview

FASCHINGBAUER

ltrace

FASCHINGBAUER

- Verwendung wie `strace` (kann auch System Calls)
- Überwacht statt System Calls Einsprungspunkte in *Shared Libraries*
- $\rightarrow$ völlig anderer Scope
- $\rightarrow$ um einiges "lauter"

Overview

FASCHINGBAUER

Valgrind

FASCHINGBAUER

Valgrind: debugging at its best (because it is so simple)

- Main target: memory errors
- Writing and reading beyond array bounds
- Usage of uninitialized memory
- Double `free`/`delete`
- Memory leaks

Drawback: considerable execution slowdown $\rightarrow$

- Race conditions not easily debugged
- Multithreading is hard generally
- Larger programs are not easily emulated $\rightarrow$ smaller test suites that are regularly checked with `valgrind`

Valgrind in Action (1)

FASCHINGBAUER

There are bugs that cannot be found because they

- *almost* never occur
- *almost* never are visible
- Cannot be reproduced in tests programs
- ...

Find the Bug!

```
#include <stdlib.h>
void main(void)
{
    char *bug = malloc(64);
    bug[64] = '\0';
}
```

Valgrind in Action (2)

FASCHINGBAUER

valgrind at Bug Search

```
$ valgrind ./a.out
...
Invalid write of size 1
   at 0x400552: main (array-bounds-write.c:5)
 Address 0x51bb072 is 0 bytes after a block of size 50
   at 0x4C28C6D: malloc (vg_replace_malloc.c:236)
   by 0x400545: main (array-bounds-write.c:4)
...
```

Valgrind in Action (3)

FASCHINGBAUER

Memory leak

```
$ valgrind --leak-check=full ./a.out
...
50 bytes in 1 blocks are definitely lost in loss rec..
   at 0x4C28C6D: malloc (vg_replace_malloc.c:236)
   by 0x400545: main (array-bounds-write.c:4)
...
```

→ *very* helpful!

Valgrind: more ...

FASCHINGBAUER

Uncovers many more types of errors:

- Usage of uninitialized variables
- *De*allocation errors (`free`/`delete`/`delete[]`)
- Erroneous system call usage
- ...

More information:

- `valgrind.org`
- `man valgrind` (as always)

Overview

FASCHINGBAUER

Overview

FASCHINGBAUER

Booting - Overview

FASCHINGBAUER

Many terms → confusion:

- Root Filesystem
- Root Directory
- Kernel Commandline
- Userspace
- initrd
- initramfs
- OS Image
- `init`

Root Filesystem

FASCHINGBAUER

- Definition: the *Root Filesystem* is the filesystem where the first program is
- "Userspace is born"
- Trditionally called `init` (but can be anything)
- Problem: how does the kernel know where the root filesystem is?
- Kernel commandline: for example, `root=/dev/sda1`, or `root=/dev/mtdblock3`
- `/sbin/init` if not otherwise specified. Explicit: `init=/my/init`
- Driver for root filesystem has to be built into kernel image
  - Modules are loaded from userspace

→ Kernel *mounts* root filesystem as specified on kernel commandline (visible in `/proc/cmdline`)

Root Filesystem, More Complex

FASCHINGBAUER

Problem: a filesystem's parameters aren't always as simple as `/dev/sda1`

...

- Network Filesystem (NFS). Historically implemented in the kernel.
- Encrypted partition → many parameters (algorithm, pass phrase, ...)
- Logical Volume Manager (LVM)
- ...

→ Not easily governed via the kernel commandline
→ **Solution:** "Early Userspace"

Overview

FASCHINGBAUER

The `mount` Command

FASCHINGBAUER

Hierarchy of Unix systems is transparently extensible (*26 drive letters? What the ...?!?*)

- "Mounting" a filesystem on a *mount point*
- Hierarchy is *transparently* composed of multiple filesystems
- Filesystem is contained in a *block device*

Mounting, e.g.:

```
# mount /dev/sdb1 /mnt/usb-stick
# mount /dev/mmcblk0p3 /home
```

The `mkfs` Command

FASCHINGBAUER

How are filesystems created? (Doze: how are partitions *formatted*?)

- Thousands of different filesystems: `ext2`, `ext3`, `xfs`, `btrfs`, ...
- Every filesystem has a different *format*
- → Filesystem specific `mkfs` programs; z.B. `mkfs.ext2`
- Flash filesystems are different
  - Operate directly in flash memory → no block device involved

`mkfs`

```
# mkfs.ext2 /dev/sdb1
```

Loop Mounts — Filesystem in a File (1)

FASCHINGBAUER

Question: if `/dev/sda1` looks like a file, why shouldn't a real file contain a file system?

Answer: who said it cannot?

- `mkfs` can operate on files (everything is a file, right?)
- *But*: a file is not a block device → "loop" mount

Step one: create empty file

```
# dd if=/dev/zero of=./my-image bs=4096 count=1024
```

Loop Mounts — Filesystem in a File (2)

FASCHINGBAUER

Step two: filesystem into file

```
# mkfs.ext2 ./my-image
mke2fs 1.41.14 (22-Dec-2010)
./my-image is not a block special device.
Proceed anyway? (y,n) y
```

`man mkfs.ext2` → `-F` to suppress annoying question

Check: file type?

```
# file ./my-image
./my-image: Linux rev 1.0 ext2 filesystem data, ...
```

Loop Mounts — Filesystem in a File (3)

FASCHINGBAUER

Mounting the *image* on a *mount point* ...

Loop-mounting `my-image`

```
# mkdir ./my-mountpoint
# mount -o loop ./my-image ./my-mountpoint
# ls ./my-mountpoint/
lost+found
```

Loop Mounts — Filesystem in a File (4)

FASCHINGBAUER

Image is now mounted → one can modify it just like any other filesystem

```
# cp -r ~jfasch ./my-mountpoint
# umount ./my-mountpoint
```

Overview

FASCHINGBAUER

The Root Directory

FASCHINGBAUER

The root directory is special:

- Absolute paths (e.g. `/bin/bash`) do start there
- There are no entries above it
- → Cannot escape → "Jail"

Exact definition:

- "Root directory" is a process attribute → each process can have its own root directory
- Path lookup starts there
- A process's "root directory" attribute is inherited → child processes have the same root as its parent

→ not so special at all!

Changing Root Directory — `chroot` (System Call

FASCHINGBAUER

System Call `chroot` (→ `man -s 2 chroot`)

- Changes path lookup for the calling process (*and does nothing else*)
- Current Working Directory (CWD) remains the same
- Open files remain open
- → relatively useless on its own

Changing Root Directory — `chroot` (Command)

FASCHINGBAUER

Command `chroot` (→ `man chroot`)

- Shell Command
- Combines `chroot()` with execution of a program
- Program must exist in new root
- All prerequisites (shared libraries, ...) must exist in new root

→ "Chroot Jail"

chroot: Demo Time

FASCHINGBAUER

... working environment with /bin/bash ...

chroot: Use Cases

FASCHINGBAUER

- Environment for services that are not trustworthy (better yet: containers, virtual machines)
- Build environment for other systems (building for Ubuntu on a Fedora system for example)
- "Boot-through": booting into a temporary RAM filesystem (*initramfs*), load drivers from there (NFS, encryption, whatever), mount *real root*, and then boot into the now-mounted *real root*

Overview

FASCHINGBAUER

Bind Mounts

FASCHINGBAUER

- Chroot-Jail is *a jail*
- → Symbolic links to the outer world don't work

→ **Bind Mounts**

- Mount files and directories, rather than device nodes
- → Mount points can be files and directories

Bind Mounts: Demo Time

FASCHINGBAUER

Bind Mount: example

```
# mkdir -p ./my-mountpoint/home/jfasch
# mount -o bind /home/jfasch ./my-mountpoint/home/jfasch
```

...

Move Mounts

FASCHINGBAUER

To move mount points cries for trouble (`umount` is confused ...)

Clean method:

Move mounts

```
# mkdir old-mountpoint new-mountpoint
# mount /dev/sda1 old-mountpoint
# mount --move old-mountpoint new-mountpoint}
```

Use: Initramfs is a typical example

- Main task: prepare real/final root filesystem
- Temporarily mounted somewhere
- At the time of switching ($\rightarrow$ `chroot`) into the real root filesystem, `procfs` und `sysfs` are moved there

Overview

FASCHINGBAUER

Late vs. Early Userspace

FASCHINGBAUER

"Late Userspace"

- Kernel has to do *a lot* to make root filesystem available
- Hardware initialization (SATA, MTD, ...)
- Mounting the filesystem, applying the right parameters
  - Parameters usually passed via *kernel commandline*
- $\rightarrow$ inflexible!

Doing complicated things does not belong in the kernel $\rightarrow$ "Early Userspace"

RAM Filesystem

FASCHINGBAUER

`ramfs` - **RAM Filesystem**

- Simple filesystem in RAM
- Grows and shrinks with content

Elder brother, the fat and dumb *ramdisk* ...

- Fixed sized block device in RAM
- Contains a real file system

Initial RAM Filesystem — `initramfs`

FASCHINGBAUER

- Kernel has always a `cpio` archive built-in
- Empty by default
- During boot: unpacked into a RAM filesystem → `initramfs`
- If the filesystem contains `/init` → done. `/init` (PID 1) takes control over booting.
- Else → as before, `root=/dev/sda1` etc.

Initial RAM Filesystem — Demo Time

FASCHINGBAUER

- `CONFIG_INITRAMFS_SOURCE` (“General setup”)
- Don’t forget `console 5 1`

Overview

FASCHINGBAUER

Overview

FASCHINGBAUER

Network File System

FASCHINGBAUER

Network File System (NFS): mounten von Directories auf Remote Rechnern

- File System nicht wie traditionell in einem Block Device
- *exportiert* von einem Server
- "Shares": gemeinsame Dateien auf zentralem Server
- Home-Directory übers Netz
- Rootfilesystem über NFS $\rightarrow$ Embedded-Entwicklung

Client: Mounten

FASCHINGBAUER

Explizites Mount Command

```
# mount -t nfs -o nolock \
      server.home.com:/files /net-files
```

Über `/etc/fstab`

```
server.home.com:/files /net-files nfs nolock 0 0
```

Client: Mount-Optionen

FASCHINGBAUER

`mount.nfs` versteht viele Optionen ($\rightarrow$ `man 5 nfs`)

- `nolock`: File-Locking über NLM abschalten (nötig, falls Server das nicht kann). Obsolet in NFS Version 4.
- `nfsvers=2`: NFS Version explizit, anstatt es Client/Server Handshake zu überlassen.
- `proto=udp`: UDP als Trägerprotokoll (statt default TCP).

Client: Root Filesystem über NFS

FASCHINGBAUER

Linux Kernel kann Root Filesystem über NFS mounten.

- `root=/dev/nfs`: Aktivieren des Kernel-Features "Root über NFS"
- `nfsroot=192.168.1.42:/the/rootfs:nolock`: analog `mount` Commandline
- `ip=dhcp`: IP Autokonfiguration via DHCP. Ohne DHCP, explizit: bitte nachlesen in `Documentation/filesystems/nfs/nfsroot.txt` im Linux Kernel Source

Server: Exportieren

FASCHINGBAUER

`/etc/exports`

```
/files *.home.com(rw,no_root_squash,no_subtree_check)
```

- *Root Squashing*: Mount Requests von Remote User `root` werden auf `nobody` gemapped → Problem, wenn Root Filesystem über NFS bezogen wird.
- *Subtree Check*: Teure Checks am Server ("Ist File unter exportiertem Directory?") → normalerweise abgeschalten.

```
# exportfs -a
```

Overview

FASCHINGBAUER

Overview

FASCHINGBAUER

SSH: Überblick (1)

FASCHINGBAUER

"Secure Shell": *sicheres* Login auf andere Rechner

- *Unverschlüsselte* Programme/Protokolle wie `rsh`, `rlogin`, `telnet`, ...
- → Passwörter auf der Leitung lesbar!
- → so leicht muss man's der NSA nicht machen!
- → SSH

SSH: Überblick (2)

FASCHINGBAUER

- SSH ist ein Protokoll
- Hauptaufgabe: Authentifizierung und Verschlüsselung
- Hauptbenutzung: *Remote Login*
- Weiter benutzt als:
  - Kopierprogramm zwischen Rechnern (`scp`)
  - File Transfer (`sftp`) — fast benutzbar als Network File System (→ `sshfs`/FUSE)
  - Port Forwarding
  - Tunneling (a.k.a VPN)
  - ...

SSH: Login (1)

FASCHINGBAUER

Einloggen als User `jfasch`:

```
$ ssh jfasch@home.com
```

Einloggen unter selbem Usernamen wie lokal:

```
$ ssh home.com
```

Einloggen und Ausführen des Commands durch die Login-Shell:

```
$ ssh jfasch@home.com ls -l
$ ssh jfasch@home.com 'ps -efl | grep sshd'
```

SSH: Exitstatus, Stdin, Stdout

FASCHINGBAUER

Exitstatus des Remote Commands wird weitergegeben:

```
$ ssh jfasch@home.com rm -f /etc/passwd
rm: unable to remove ‘/etc/passwd’: Permission denied
$ echo $?
1
```

Standard Input, Output (und Error) bleiben wie sie sind:

```
{ echo hallo; echo hello; } | \
    ssh jfasch@home.com cat | \
    wc -l
```

Achtung: nur der mittlere Teil (`ssh cat`) der Pipe läuft remote; `wc -l` läuft *lokal*

SSH: Public-Key Authentifizierung (1)

FASCHINGBAUER

Password-Authentifizierung ist unsicher:

- Man-in-the-Middle
- Brute-Force Attacken zum Passwort-Erraten

... und unbequem: Login nur interaktiv möglich $\rightarrow$ Schlüsselpaar

- *Public Key* am Server
- *Private Key* lokal. Achtung: den sollte man nicht weitergeben!

SSH: Public-Key Authentifizierung (2)

FASCHINGBAUER

Und so gehts ...

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
...
```

- *Public Key* `~/.ssh/id_rsa.pub` wird auf den Server transferiert und an `~/.ssh/authorized_keys` angehängt.
- *Private Key* `~/.ssh/id_rsa` bleibt lokal.

SSH: Public-Key Authentifizierung (3)

FASCHINGBAUER

Zu beachten:

- Passphrase: zusätzlicher Schutz, sollte man den Private Key verlieren
- → aber dadurch nur interaktives Login möglich
- Passphrase weglassen (2x leer) → Eingabe entfällt → nichtinteraktiv
- `ssh-agent`: einmalige Passphrase-Eingabe durch `ssh-add`
  - merkt sich Passphrase (nicht persistent, sondern im Memory)
  - wird von `ssh` darum gebeten → nicht-interaktiv *mit* Passphrase
- *Permissions* von `~/.ssh` und den enthaltenen Files wichtig!
  Ansonsten weigert sich `ssh`

Overview

FASCHINGBAUER

scp

FASCHINGBAUER

`scp`: **“Secure Copy”** → **wie** `cp`**, nur über SSH** Protokoll

Einzelnes File hinüber:

```
$ scp ~/.bashrc jfasch@home.com:
# oder
$ scp ~/.bashrc jfasch@home.com:~
```

Einzelnes File herüber:

```
$ scp jfasch@home.com:.bashrc ~/.bashrc
```

Rekursiv (wie `cp -r`**)**:

```
$ scp -r ~/Downloads jfasch@home.com:
```

Overview

FASCHINGBAUER

Port Forwarding — Das Problem (1)

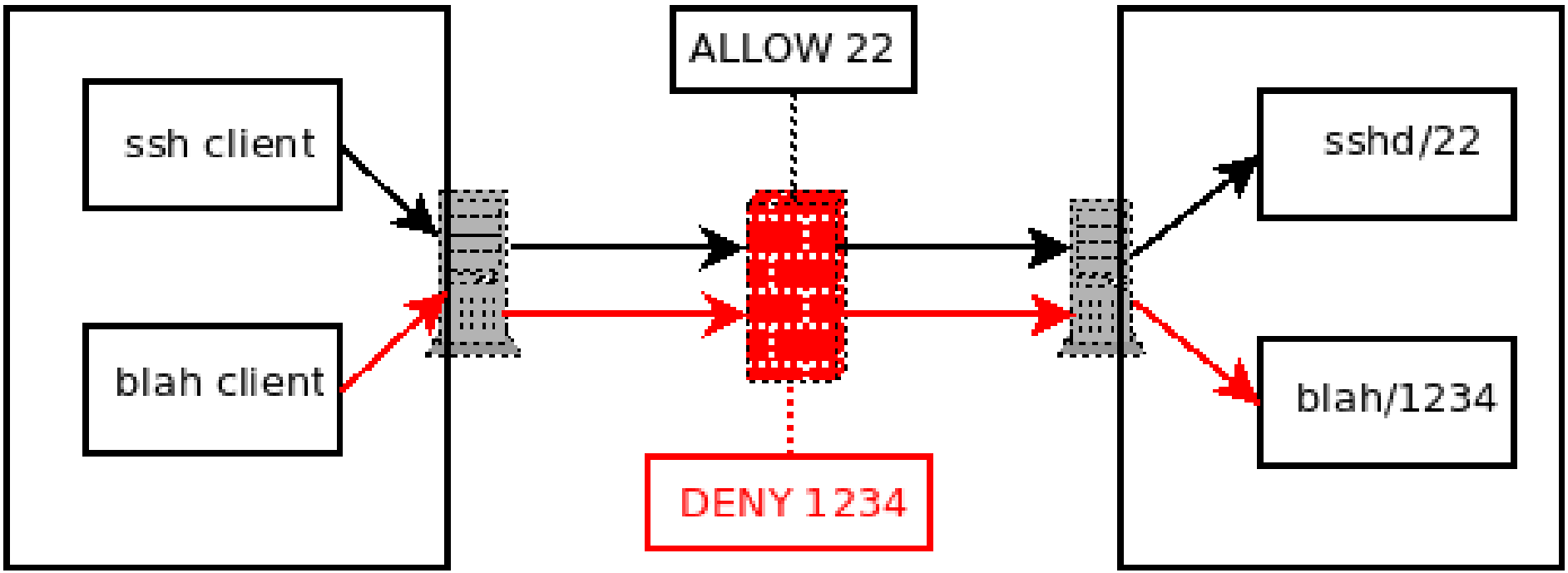
FASCHINGBAUER

Problem:

1. *Remote* Port ist unerreichbar. Auf dem Remote Rechner läuft ein Programm (ein "Server"), das auf den Port 1234 hört. Das Protokoll ist in Klartext, und der dortige Netzwerkadministrator weigert sich (recht hat er), diesen Port an der Firewall freizuschalten.
2. Ich habe SSH Zugriff auf den Rechner, da mir persönlich vertraut wird.

Port Forwarding — Das Problem (2)

FASCHINGBAUER



Port Forwarding — Die Lösung (1)

FASCHINGBAUER

Hmmm ...

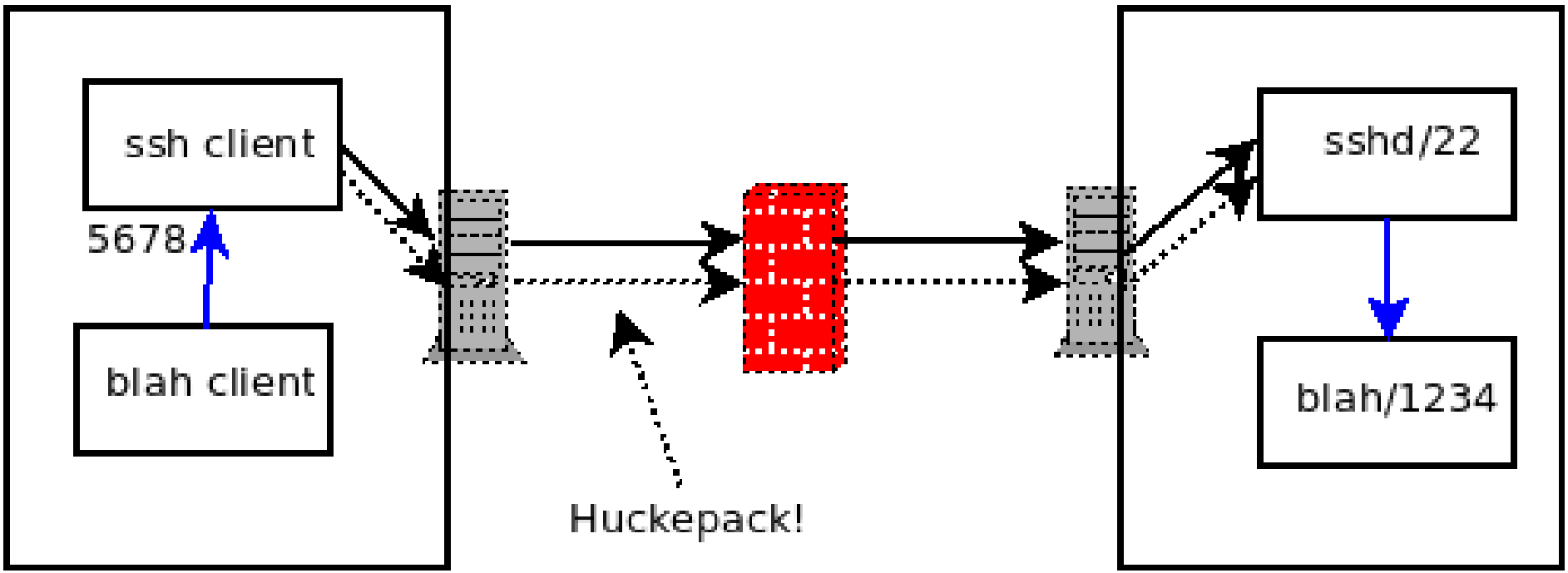
- Mir wird vertraut → ich habe Zugriff auf Port 22/ssh.
- Daraus folgt: ich darf auf dem Rechner einloggen
- Daraus folgt: ich darf dort alles machen, wozu ich berechtigt bin
- Daraus folgt: ich darf mich einloggen und kann dort Port 1234 erreichen

Hmmm ...

Dann ist es kein Securityrisiko für den Administrator, wenn ich das ganze automatisiere!

Port Forwarding — Die Lösung (2)

FASCHINGBAUER



Portforwarding — Die Commandline

FASCHINGBAUER

Beim Login den "Tunnel" herstellen

```
$ ssh -L 5678:127.0.0.1:1234 jfasch@home.com
```

- `blah` Client verbindet zu 5678 *lokal*
- SSH Server verbindet *remote* zum *dortigen* lokalen (127.0.0.1) Port 1234

Overview

FASCHINGBAUER

Overview

FASCHINGBAUER

Package Management: Probleme

FASCHINGBAUER

Warum Package Management?

Probleme:

- Pakete sollen möglichst einfach installiert werden können
- Pakete sollen deinstalliert werden können
- Systemintegrität soll gewahrt bleiben
- Updates sollen möglich sein
- Keine duplizierten Abhängigkeiten, zum Unterschied von Android's `.apk` (und natürlich Windows)

Package Management: Lösungen

FASCHINGBAUER

Ein Problem, viele Lösungen (Definition von Open Source)

- Redhat's RPM: RHEL, Fedora, SUSE, Mandriva, ...)
- Debian's dpkg: Debian, Ubuntu, ...)
- Gentoo's Portage: baut von Source
- opkg: Embedded-Systeme
- Yocto: Embedded-Systeme, baut von Source. "Meta-Distro": baut Pakete für z.B. RPM, Debian, opkg, ...
- u.v.a.m.

Pakete: `.deb` Files

FASCHINGBAUER

Paket

- Ein Paket wird installiert aus genau einer Datei
- Name z.B. `unzip-6.0-8.deb`
- → "Das Paket `unzip` in der Version 6.0-8

Enthaltene Informationen

- Daten (Programme, Libraries, sonstige Dateien)
- Metainformation
  - Abhängigkeiten: was muss noch installiert sein (in welcher Version), damit dieses Paket funktioniert?
  - Trigger: vor Installation, nach Installation, vor Removal, ...

dpkg: Pakete, Paketdatenbank

FASCHINGBAUER

Low-Level Tool: dpkg

- Versteht .deb Files
- Installiert diese, und pflegt dabei die Paketdatenbank
- Deinstalliert diese, natürlich mit Datenbank
- Installiert nur Paket selbst, keine Abhängigkeiten

→ dpkg *implementiert* Debian's Package Management

dpkg: Information

FASCHINGBAUER

dpkg: .deb Files

`--info unzip-6.0-8.deb`	Metainformation
`--contents unzip-6.0-8.deb`	Files

dpkg: Paketdatenbank

`--list`	Kurzinfo zu allen installierten Pakete
`--list unzip`	Kurzinfo zu `unzip`
`--listfiles unzip`	Alle Files, die zu `unzip` gehören
`--status unzip`	Mehr Info zu `unzip`
`--search /usr/bin/unzip`	Zu welchem Paket gehört ...
`which unzip`	Wie ist der absolute Pfad zu `unzip`

dpkg: Pakete verwalten

FASCHINGBAUER

dpkg: Pakete verwalten

`--install unzip-6.0-8.deb`	Paket von `.deb` installieren
`--remove unzip`	Paket `unzip` deinstallieren
`--purge unzip`	Paket `unzip` *rauskratzen*

- Achtung: `--purge` löscht Configfiles, die unter Umständen per Hand editiert wurden.
- **Klar?** Installiert wird ein *`.deb` File*, deinstalliert wird das *Paket*.

dpkg: Was fehlt?

FASCHINGBAUER

Datenbank von installierten Packages ist Grundvoraussetzung für ein gesundes System. Aber es fehlen wichtige Features:

- Installieren eines Package *samt seiner Abhängigkeiten* → *Abhängigkeitsgraph*
- Löschen eines Package - und aller seiner Abhängigkeiten, die von sonst niemandem mehr gebraucht werden.
- Liste von *verfügbaren* Packages
- Automatisches `.deb` Download → *Package Repositories*

→ Advanced Package Tool - APT

APT: Advanced Package Tool

FASCHINGBAUER

- Kein Programm, nur gemeinsamer Code
- Mehrere Frontends (`apt-get`, `aptitude`, `synaptic`)
- *Cache*: Datenbank des "grossen Bildes"
- Konfiguration in `/etc/apt/`, z.B. `/etc/apt/sources.list`

apt-get

FASCHINGBAUER

apt-get: Low-level Commandline Tool

`update`	Paketinformationen aktualisieren
`upgrade`	Paketupdates *installieren*
`install unzip`	Paket `unzip` installieren
`remove unzip`	Paket `unzip` deinstallieren
`autoremove`	nicht benötigte Pakete deinstallieren

- `install` installiert Paket *und alle von diesem benötigten Pakete*
- *Nicht benötigte Pakete*: der "Benötiger" wurde deinstalliert

Weitere APT Frontends

FASCHINGBAUER

- `aptitude`: GUI für Arme. Textbasiertes APT-Schweizermesser
- `synaptic`: Grafisches GUI. Populär seit Ubuntu.

Overview

FASCHINGBAUER

Overview

FASCHINGBAUER

Devices

FASCHINGBAUER

- Repräsentiert durch "Character Device Nodes"
- Per Konvention und per udev in /dev

Serielle Schnittstellen am PC

```
$ ls -l /dev/ttyS*
crw-rw---- 1 root uucp 4, 64 Sep 14 17:46 /dev/ttyS0
...
```

- Können auf anderen Architekturen anders heissen (ttyS → ttyBLA)
- ttyS0 ist die erste serielle Schnittstelle (COM1), ttyS1 die zweite, etc.
- USB/Serial Konverter heissen ttyUSB0, ttyUSB1, ...

Konfiguration (1)

FASCHINGBAUER

Ausgeben der Settings

```
$ stty -F /dev/ttyS0 -a
speed 115200 baud; rows 0; columns 0; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^A; ...
lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr ...
...
```

Konfiguration (2)

FASCHINGBAUER

Verändern der Settings

```
$ stty -F /dev/ttyS0 speed 57600
115200
```

→ Gibt vorherigen Wert aus

Konfiguration (3)

FASCHINGBAUER

setserial — Lowlevel Details

```
$ setserial -g /dev/ttyS0 -a
/dev/ttyS0, Line 0, UART: unknown, Port: 0x03f8, IRQ: 4
Baud_base: 115200, close_delay: 50, divisor: 0
closing_wait: 3000
Flags: spd_normal skip_test auto_irq
$ setserial /dev/ttyS0 low_latency
```

Overview

FASCHINGBAUER

Overview

FASCHINGBAUER

Commandline aus Sicht der Shell

FASCHINGBAUER

Ein typisches Command

```
$ ls -al /tmp
```

- Shell sucht ein Programm `ls` entlang des Pfades (`$PATH`)
  - ... oder benutzt z.B. `execvp`
- Führt es mittels `execve` aus
- Argumente:
  - Index 0: Programmname, ‘‘`ls`’’. *Wichtig*: das nur eine Information an das Programm!
  - Index 1: ‘‘`-l`’’
  - Index 2: ‘‘`/tmp`’’

Commandline aus Sicht von evecvp()

FASCHINGBAUER

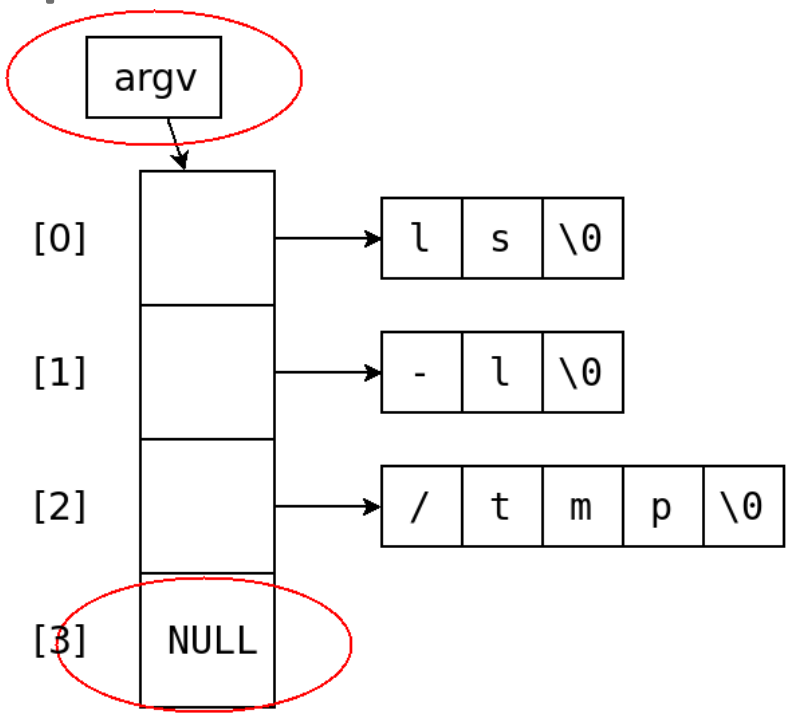
Exec Aufruf

```
// man 3 exec:
int execvp(
  const char *file,
  char *const argv[]);

char *argv[4];
argv[0] = "ls";
argv[1] = "-l";
argv[2] = "/tmp";
argv[3] = 0; // "Null-Pointer"

int rv = execvp("ls", argv);
```

Speichersituation:

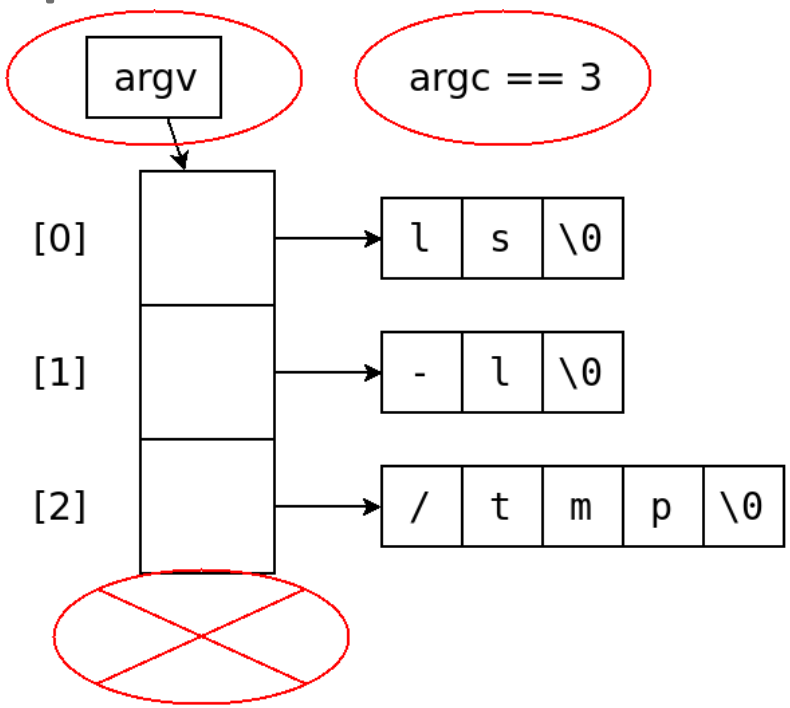


Commandline aus Sicht von `main()`

FASCHINGBAUER

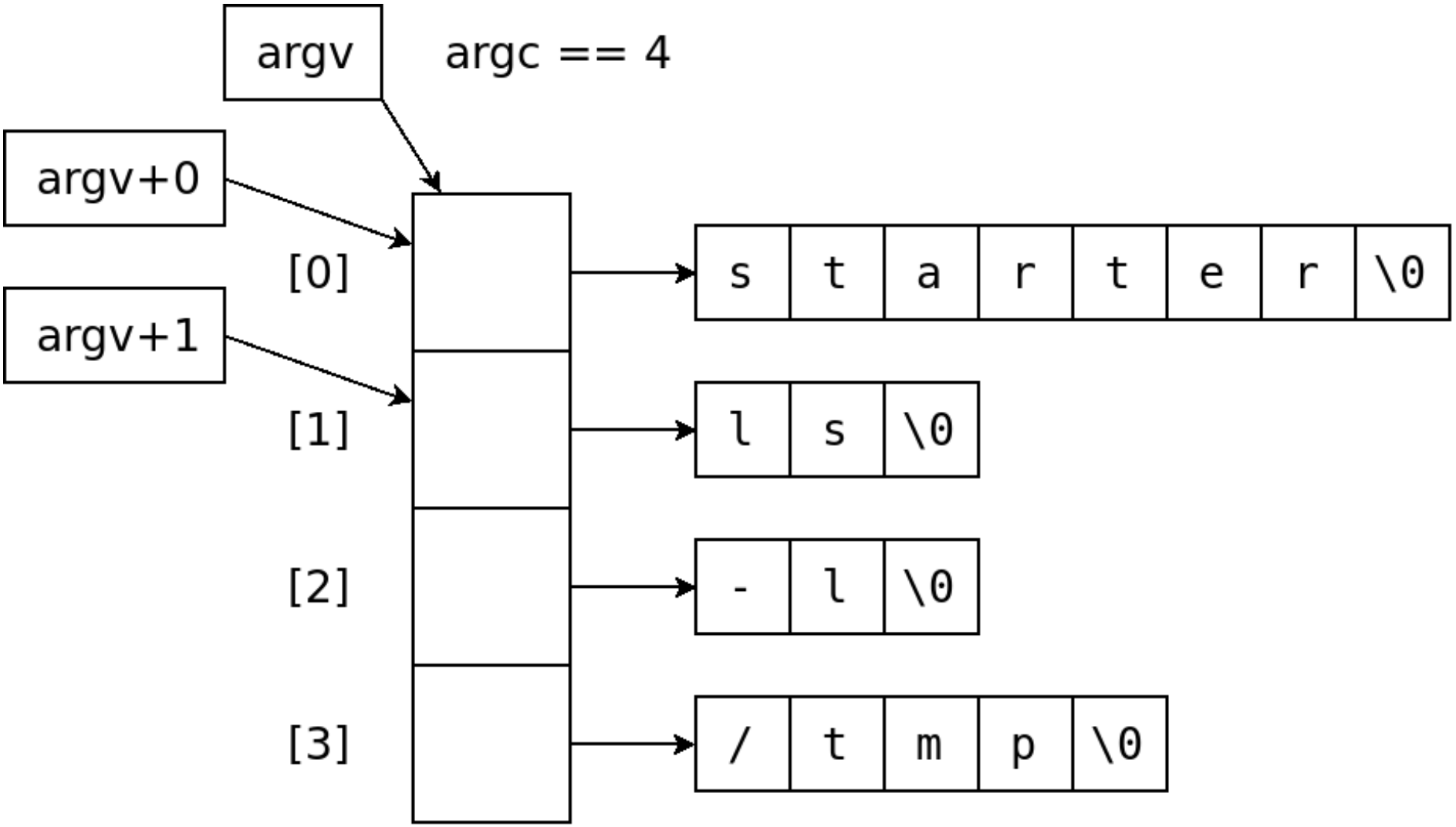
```
main()
int main(
  int argc,
  char *argv[])
{
  // ...
  return exitstatus;
}
```

Speichersituation:



argv unterliegt normaler Pointerarithmetik

FASCHINGBAUER



Overview

FASCHINGBAUER

Anhang

FASCHINGBAUER